LENORA COLLEGE OF ENGINEERING

Rampachodavaram
E.G. Dist., A.P.

CERTIFICATE

Roll No.

This is to Certify that _____

Student of B. Tech _____ in Lenora College of Engineering. Rampachodavaram, E.G. Dt., He/She has Successfully complete His/ Her Record work in _____

during the academic year 20 - 20

Examine :

Internal :

Externam :

Lab Incharge :

Head of the Department

LENORA COLLEGE OF ENGINEERING

RAMPACHODAVARAM

M.Tech - C.S.E. – R19 - MACHINE LEARNING WITH PYTHON LAB

EXPERIMENT 1:

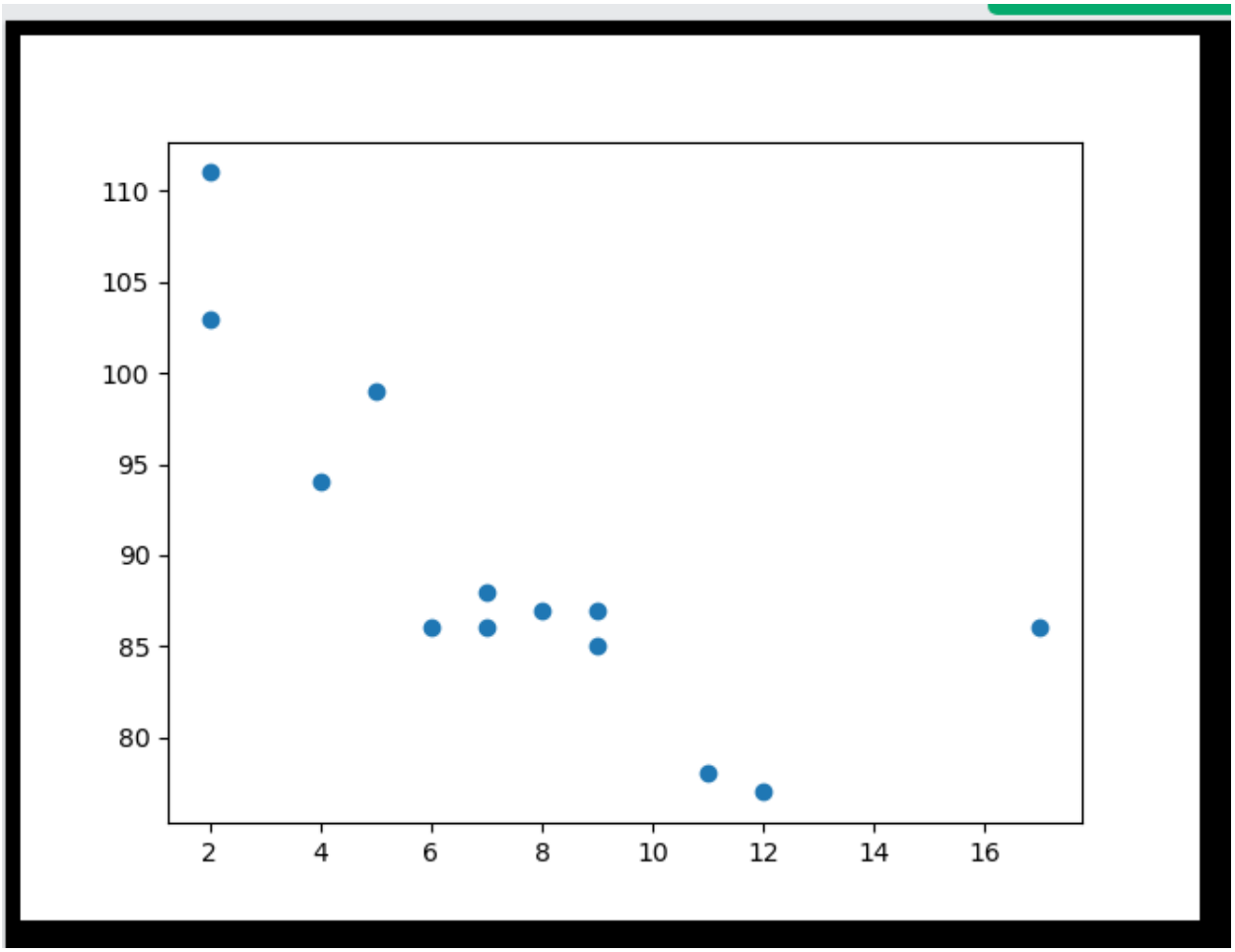
Exercises to solve the real-world problems using the following machine learning methods:

1. Linear regression
2. Logistic regression

1. Linear regression

```
#Three lines to make our compiler able to draw:

import sys

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]

y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)

plt.show()

#Two  lines to make our compiler able to draw:

plt.savefig(sys.stdout.buffer)

sys.stdout.flush()
```

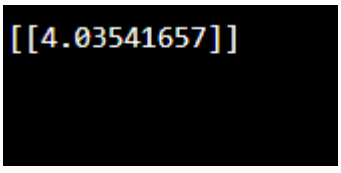
Output:



2. Logistic regression

```
import numpy

from sklearn import linear_model


X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)

y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])


logr = linear_model.LogisticRegression()

logr.fit(X,y)


log_odds = logr.coef_

odds = numpy.exp(log_odds)


print(odds)
```

Output:



Experiment 2:

Write a program to implement support vector machines.

```
import pandas as pd
import math

heart = pd.read_csv("heart_disease")

nrows = math.floor(heart.shape[0] * 0.8)

training = heart.loc[:nrows]
test = heart.loc[nrows:]
from sklearn.svm import SVC

model = SVC()
model.fit(training[["age", "chol"]], training["present"])
predictions = model.predict(test[["age", "chol"]])

accuracy = sum(test["present"] == predictions) / test.shape[0]
```

output:

```
accuracy
0.4666666666666667
```

Experiment 3:

Exploratory data analysis for classification using pandas and matplotlib.

We need to install Pandas, NumPy, Matplotlib and Seaborn libraries in python to proceed further.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings as wr

wr.filterwarnings('ignore')

df = pd.read_csv("/content/WineQT.csv")

print(df.head())

df.shape

df.info()

df.describe()

df.columns.tolist()

df.isnull().sum()

df.nunique()

quality_counts = df['quality'].value_counts()


plt.figure(figsize=(8, 6))

plt.bar(quality_counts.index, quality_counts, color='deeppink')

plt.title('Count Plot of Quality')

plt.xlabel('Quality')

plt.ylabel('Count')

plt.show()
```

Output:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	\
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	
3	11.2	0.28	0.56	1.9	0.075	
4	7.4	0.70	0.00	1.9	0.076	

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	\
0	11.0	34.0	0.9978	3.51	0.56	
1	25.0	67.0	0.9968	3.20	0.68	
2	15.0	54.0	0.9970	3.26	0.65	
3	17.0	60.0	0.9980	3.16	0.58	
4	11.0	34.0	0.9978	3.51	0.56	

	alcohol	quality	Id
0	9.4	5	0
1	9.8	5	1
2	9.8	5	2
3	9.8	6	3
4	9.4	5	4

Output:

	0
fixed acidity	91
volatile acidity	135
citric acid	77
residual sugar	80
chlorides	131
free sulfur dioxide	53
total sulfur dioxide	138
density	388
pH	87
sulphates	89
alcohol	61
quality	6
Id	1143

Output:

	0
fixed acidity	0
volatile acidity	0
citric acid	0
residual sugar	0
chlorides	0
free sulfur dioxide	0
total sulfur dioxide	0
density	0
pH	0
sulphates	0
alcohol	0
quality	0
Id	0

dtype: int64

Output:

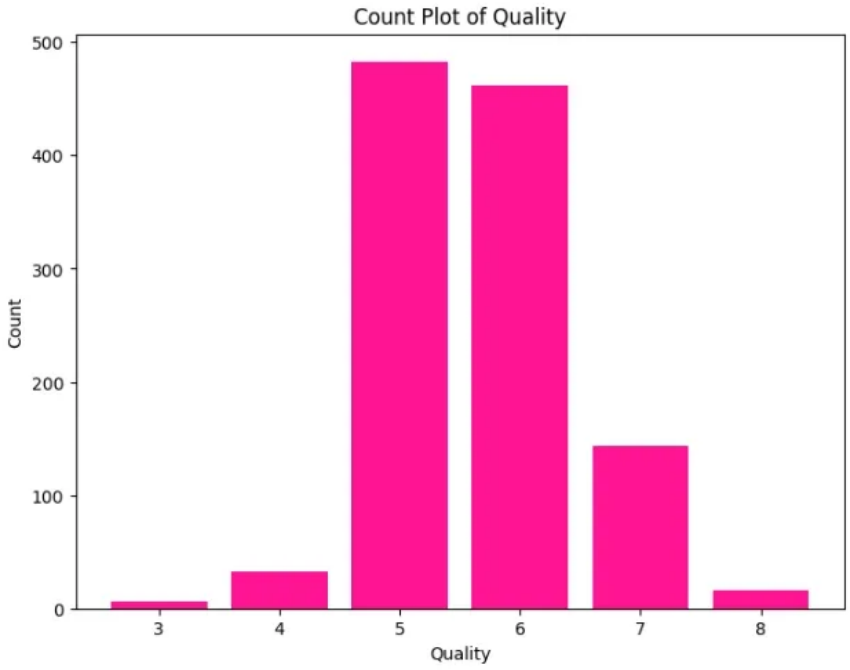
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol',
 'quality',
 'Id']
```

Output:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45.914698	0.996730	3.311015
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.782130	0.001925	0.156664
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000	0.995570	3.205000
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000	0.996680	3.310000
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000	0.997845	3.400000
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289.000000	1.003690	4.010000

describe()

Output:



Experiment 4:

Implement a program for bias, variance and cross validation.

```
# Import the necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from mlxtend.evaluate import bias_variance_decomp
import warnings
warnings.filterwarnings('ignore')

# Load the dataset
X, y = load_iris(return_X_y=True)

# Split train and test dataset
X_train, X_test,\
   y_train, y_test = train_test_split(X, y,
                                      test_size=0.25,
                                      random_state=23,
                                      shuffle=True,
                                      stratify=y)

# Build the classification model
tree = DecisionTreeClassifier(random_state=123)
clf = BaggingClassifier(base_estimator=tree,
                        n_estimators=50,
```

```
                    random_state=23)


# Bias variance decompositions
avg_expected_loss, avg_bias, \
    avg_var = bias_variance_decomp(clf,
                                   X_train, y_train,
                                   X_test, y_test,
                                   loss='0-1_loss',
                                   random_seed=23)
# Print the value
print('Average expected loss: %.2f' % avg_expected_loss)
print('Average bias: %.2f' % avg_bias)
print('Average variance: %.2f' % avg_var)
```

Output:

Average expected loss: 0.06
Average bias: 0.05
Average variance: 0.02

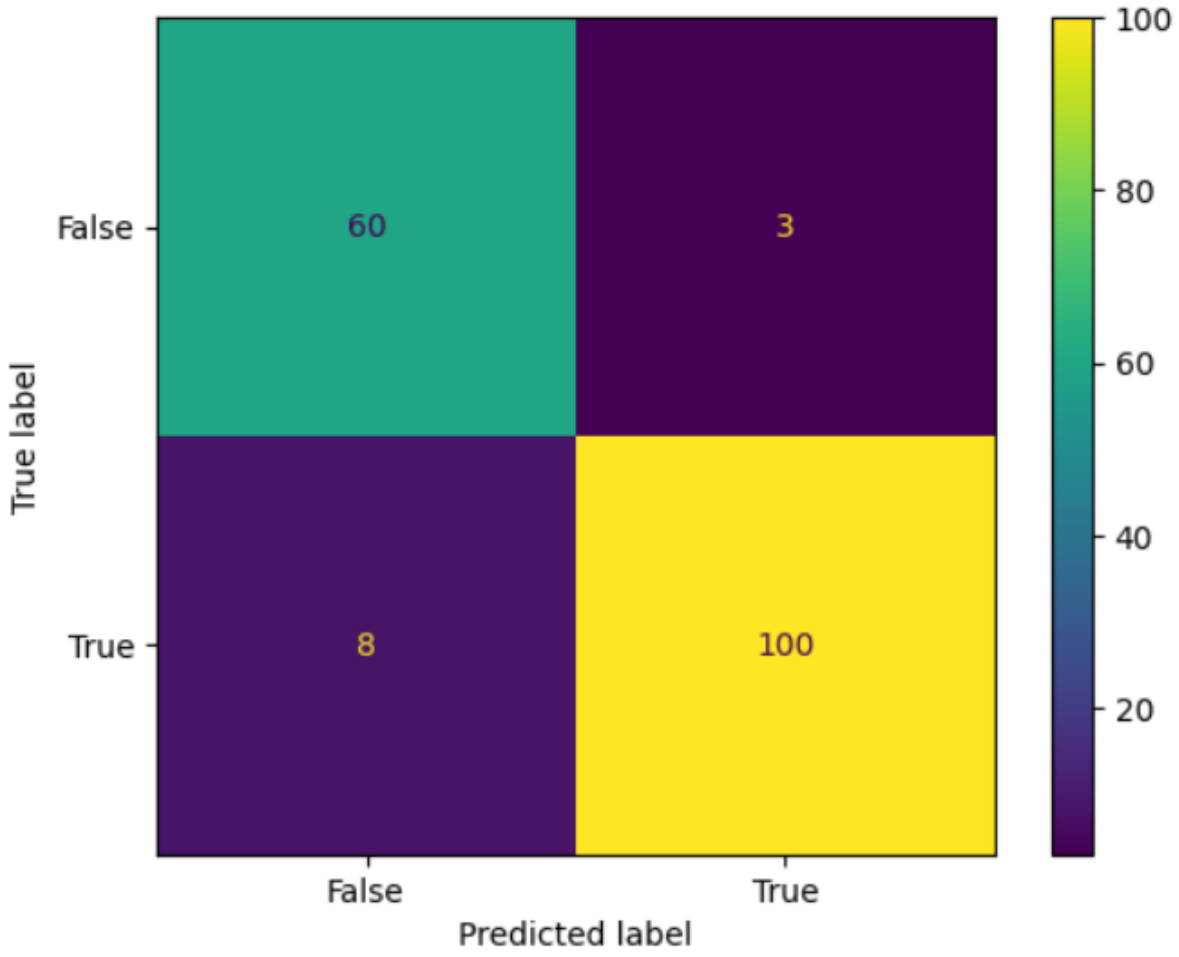
Experiment 5:

Write a program to simulate a perception network for pattern classification and function approximation.

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target
print(X.shape)
print(y.shape)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
# Create a Perceptron model
clf = Perceptron(max_iter=1000, eta0=0.1)
# Train the model
clf.fit(X_train, y_train)
# Make predictions
y_pred = clf.predict(X_test)
```

```
print(y_pred)

# Evaluate accuracy

accuracy = accuracy_score(y_test, y_pred)

print(&amp;quot;Accuracy:&amp;quot;, accuracy)

# Calculate confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = conf_matrix, display_labels =
[False, True])


cm_display.plot()

plt.show()
```

Output:



Experiment 6:

Write a program to demonstrate the working of the decision tree based ID3algorithm. Use an appropriate data set for bulking the decision tree and apply this knowledge to classify the new sample.

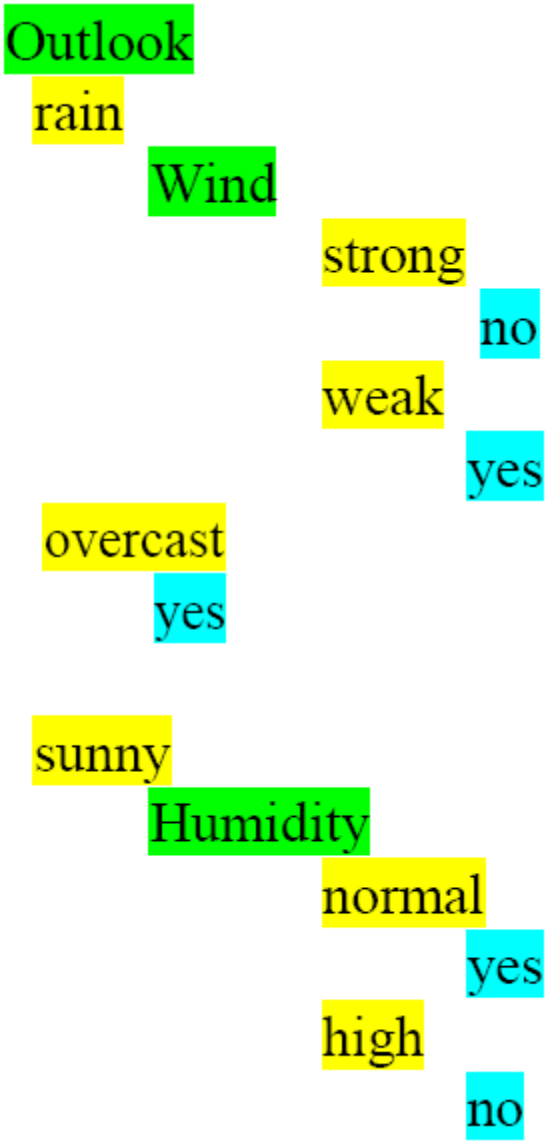
```
import pandas as pd

import math

import numpy as np


data = pd.read_csv("Dataset/4-dataset.csv")

features = [feat for feat in data]

features.remove("answer")

class Node:

    def __init__(self):

        self.children = []

        self.value = ""

        self.isLeaf = False

        self.pred = ""

def entropy(examples):

    pos = 0.0

    neg = 0.0

    for _, row in examples.iterrows():

        if row["answer"] == "yes":

            pos += 1

        else:

            neg += 1

    if pos == 0.0 or neg == 0.0:

        return 0.0
```

```
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain
def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
```

```
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)

    return root
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
```

```
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new," is:", child.pred)
                exit
            else:
                classify (child.children[0], new)
root = ID3(data, features)
print("Decision Tree is:")
printTree(root)
print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}
classify (root, new)
```

Output:


Outlook
rain
Wind
strong
no
weak
yes
overcast
yes
sunny
Humidity
normal
yes
high
no

Experiment 7:

Build an artificial neural network by implementing the back propagation algorithm and the test the same using appropriate data sets.

```
import numpy as np



class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.weights_input_hidden = np.random.randn(
            self.input_size, self.hidden_size)
        self.weights_hidden_output = np.random.randn(
            self.hidden_size, self.output_size)

        self.bias_hidden = np.zeros((1, self.hidden_size))
        self.bias_output = np.zeros((1, self.output_size))

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        return x * (1 - x)
def feedforward(self, X):
    self.hidden_activation = np.dot(
```

```
        X, self.weights_input_hidden) + self.bias_hidden
    self.hidden_output = self.sigmoid(self.hidden_activation)

    self.output_activation = np.dot(
        self.hidden_output, self.weights_hidden_output) + self.bias_output
    self.predicted_output = self.sigmoid(self.output_activation)

    return self.predicted_output
def backward(self, X, y, learning_rate):
    output_error = y - self.predicted_output
    output_delta = output_error * \
        self.sigmoid_derivative(self.predicted_output)

    hidden_error = np.dot(output_delta, self.weights_hidden_output.T)
    hidden_delta = hidden_error * self.sigmoid_derivative(self.hidden_output)

    self.weights_hidden_output += np.dot(self.hidden_output.T,
                                         output_delta) * learning_rate
    self.bias_output += np.sum(output_delta, axis=0,
                               keepdims=True) * learning_rate
    self.weights_input_hidden += np.dot(X.T, hidden_delta) * learning_rate
    self.bias_hidden += np.sum(hidden_delta, axis=0,
                               keepdims=True) * learning_rate

def train(self, X, y, epochs, learning_rate):
    for epoch in range(epochs):
        output = self.feedforward(X)
```

```
        self.backward(X, y, learning_rate)
        if epoch % 4000 == 0:
            loss = np.mean(np.square(y - output))
            print(f"Epoch {epoch}, Loss:{loss}")
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

nn = NeuralNetwork(input_size=2, hidden_size=4, output_size=1)
nn.train(X, y, epochs=10000, learning_rate=0.1)

output = nn.feedforward(X)
print("Predictions after training:")
print(output)
```

Output:

```
Epoch 0, Loss:0.27132035388864456
Epoch 4000, Loss:0.12865253014284214
Epoch 8000, Loss:0.006592119352308818
Predictions after training:
[[0.03837966]
 [0.93898139]
 [0.94188724]
 [0.07318271]]
```

Experiment 8:

Write a program to implement the naïve Bayesian classifier for iris data set. Compute the accuracy of the classifier, considering few test data sets.

```
import csv
import random
import math

def loadcsv(filename):
        lines = csv.reader(open(filename, "r"));
        dataset = list(lines)
        for i in range(len(dataset)):
    #converting strings into numbers for processing
                dataset[i] = [float(x) for x in dataset[i]]

        return dataset

def splitdataset(dataset, splitratio):
   #67% training size
        trainsize = int(len(dataset) * splitratio);
        trainset = []
        copy = list(dataset);
        while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for training data
                index = random.randrange(len(copy));
                trainset.append(copy.pop(index))
        return [trainset, copy]
```

```
def separatebyclass(dataset):
        separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
        for i in range(len(dataset)):
                vector = dataset[i]
                if (vector[-1] not in separated):
                        separated[vector[-1]] = []
                separated[vector[-1]].append(vector)
        return separated

def mean(numbers):
        return sum(numbers)/float(len(numbers))

def stdev(numbers):
        avg = mean(numbers)
        variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
        return math.sqrt(variance)

def summarize(dataset): #creates a dictionary of classes
        summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
        del summaries[-1] #excluding labels +ve or -ve
        return summaries

def summarizebyclass(dataset):
        separated = separatebyclass(dataset);
   #print(separated)
```

```
	summaries = {}
	for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
		summaries[classvalue] = summarize(instances) #summarize is used to cal to mean and std
	return summaries

def calculateprobability(x, mean, stdev):
	exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
	return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateclassprobabilities(summaries, inputvector):
	probabilities = {} # probabilities contains the all prob of all class of test data
	for classvalue, classsummaries in summaries.items():#class and attribute information as mean and sd
		probabilities[classvalue] = 1
		for i in range(len(classsummaries)):
			mean, stdev = classsummaries[i] #take mean and sd of every attribute for class 0 and 1 seperaely
			x = inputvector[i] #testvector's first attribute
			probabilities[classvalue] *= calculateprobability(x, mean, stdev);#use normal dist
	return probabilities

def predict(summaries, inputvector): #training and test data is passed
	probabilities = calculateclassprobabilities(summaries, inputvector)
	bestLabel, bestProb = None, -1
```

```
        for classvalue, probability in probabilities.items():#assigns that class which has he highest
prob
                if bestLabel is None or probability > bestProb:
                        bestProb = probability
                        bestLabel = classvalue
        return bestLabel

def getpredictions(summaries, testset):
        predictions = []
        for i in range(len(testset)):
                result = predict(summaries, testset[i])
                predictions.append(result)
        return predictions

def getaccuracy(testset, predictions):
        correct = 0
        for i in range(len(testset)):
                if testset[i][-1] == predictions[i]:
                        correct += 1
        return (correct/float(len(testset))) * 100.0

def main():
        filename = 'naivedata.csv'
        splitratio = 0.67
        dataset = loadcsv(filename);

        trainingset, testset = splitdataset(dataset, splitratio)
```

```
        print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset), len(testset)))

        # prepare model

        summaries = summarizebyclass(trainingset);

        #print(summaries)

    # test model

        predictions = getpredictions(summaries, testset) #find the predictions of test data with the training data

        accuracy = getaccuracy(testset, predictions)

        print('Accuracy of the classifier is : {0}%'.format(accuracy))


main()
```

Output

```
Split 768 rows into train=514 and test=254
```

Experiment 9:

Assuming a set of documents that need to be classified, use the naïve Bayesian classifer model to perform this task. Built in java classes/api can be used to write the program. Calculate the accuracy, precission and recall for your data set.

```
import pandas as pd

msg=pd.read_csv('naivetext1.csv',names=['message','label'])

print('The dimensions of the dataset',msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})

X=msg.messagey=msg.labelnumprint(X)

print(y)

#splitting the dataset into train and test data from

sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest=train_test_split(X,y)

print(xtest.shape)

print(xtrain.shape)

print(ytest.shape)

print(ytrain.shape)

#output of count vectoriser is a sparse matrix

from sklearn.feature_extraction.text

import CountVectorizercount_vect = CountVectorizer()

xtrain_dtm = count_vect.fit_transform(xtrain)

xtest_dtm=count_vect.transform(xtest)

print(count_vect.get_feature_names())

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())

print(df)

#tabular representation

print(xtrain_dtm)
```

```
#sparse matrix representation

# Training Naive Bayes (NB) classifier on training data

from sklearn.naive_bayes import MultinomialNB clf

= MultinomialNB().fit(xtrain_dtm,ytrain)

predicted = clf.predict(xtest_dtm)

#printing accuracy metrics

from sklearn import metricsprint('Accuracy metrics')

print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))

print('Confusion matrix')

print(metrics.confusion_matrix(ytest,predicted))

print('Recall and Precison ')

print(metrics.recall_score(ytest,predicted))

print(metrics.precision_score(ytest,predicted))

'''docs_new = ['I like this place', 'My boss is not my saviour']

X_new_counts = count_vect.transform(docs_new)predictednew = clf.predict(X_new_counts)

for doc, category in zip(docs_new, predictednew):

print('%s->%s' % (doc, msg.labelnum[category]))'''
```

I love this sandwich,pos This is an amazing place,pos

I feel very good about these beers,posThis is my best work,pos

What an awesome view,pos

I do not like this restaurant,negI am tired of this stuff,neg

I can't deal with this,neg He is my sworn enemy,negMy boss is horrible,neg

This is an awesome place,pos

I do not like the taste of this juice,negI love to dance,pos

I am sick and tired of this place,negWhat a great holiday,pos

That is a bad locality to stay,neg

We will have good fun tomorrow,posI went to my enemy's house today,neg

OUTPUT

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'boss', 'can', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'have', 'horrible', 'house', 'is', 'like', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sandwich', 'sick', 'stuff', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will', 'with', 'work']about am amazing an and awesome beers best boss can ... today \

```
0   1 0    0 0 0 0 1  0 0 0 ... 0
1   0 0    0 0 0 0 0  1 0 0 ... 0
2   0 0    1 1 0 0    0 0 0 0 ... 0
3   0 0    0 0 0 0 0  0 0 0 ... 1
4   0 0    0 0 0 0 0  0 0 0 ... 0
5   0 1    0 0 1      0 0 0 0 0 ... 0
6   0 0    0 0 0 0 0  0 0 1 ... 0
7   0 0    0 0 0 0 0  0 0 0 ... 0
8   0 1    0 0 0 0 0  0 0 0 ... 0
9   0 0    0 1 0 1 0  0 0 0 ... 0
10 0 0   0 0 0 0 0 0 0 0 ... 0
11 0 0   0 0 0   0 0 0 1 0 ... 0
12 0 0   0 1 0 1 0 0 0 0 ... 0
```

Experiment 10:

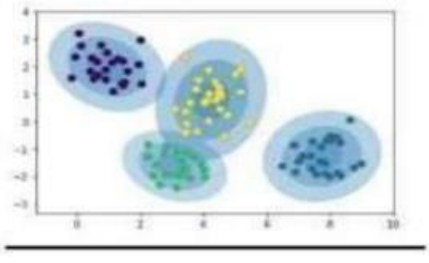
Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering usingkMeans algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets.samples_generator

import make_blobsX, y_true = make_blobs(n_samples=100, centers =

4,Cluster_std=0.60,random_state=0)

X = X[:, ::-1]

#flip axes for better plotting

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture (n_components = 4).fit(X)lables = gmm.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap="viridis");probs = gmm.predict_proba(X)

print(probs[:5].round(3))

size = 50 * probs.max(1) ** 2

# square emphasizes differences

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap="viridis", s=size);

from matplotlib.patches import Ellipse

def draw_ellipse(position, covariance, ax=None, **kwargs);

"""Draw an ellipse with a given position and covariance"""Ax

= ax or plt.gca()

# Convert covariance to principal axes

if covariance.shape ==(2,2):

U, s, Vt = np.linalg.svd(covariance)

Angle = np.degrees(np.arctan2(U[1, 0], U[0,0]))Width, height = 2 * np.sqrt(s)
```

```
else:
angle = 0
width, height = 2 * np.sqrt(covariance)
#Draw the Ellipse
for nsig in range(1,4):
ax.add_patch(Ellipse(position, nsig * width, nsig *height,angle, **kwargs))
def plot_gmm(gmm, X, label=True, ax=None):ax = ax or plt.gca()
labels = gmm.fit(X).predict(X)if label:
ax.scatter(X[:, 0], x[:, 1], c=labels, s=40, cmap="viridis", zorder=2)else:
ax.scatter(X[:, 0], x[:, 1], s=40, zorder=2)ax.axis(„equal")
w_factor = 0.2 / gmm.weights_.max()
for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):draw_ellipse(pos, covar,
alpha=w * w_factor)
gmm = GaussianMixture(n_components=4, random_state=42)plot_gmm(gmm, X)
gmm = GaussianMixture(n_components=4, covariance_type="full",random_state=42)
plot_gmm(gmm, X)
```

Output

```
[[1 ,0, 0, 0]
 [0 ,0, 1, 0]
 [1 ,0, 0, 0]
 [1 ,0, 0, 0]
[1 ,0, 0, 0]]
```



K MEANS :

```
from sklearn.cluster import KMeans

#from sklearn import metricsimport numpy as np

import matplotlib.pyplot as plt

import pandas as pd

data=pd.read_csv("kmeansdata.csv")

df1=pd.DataFrame(data)

print(df1)

f1 = df1['Distance_Feature'].valuesf2 = df1['Speeding_Feature'].values

X=np.matrix(list(zip(f1,f2)))plt.plot()

plt.xlim([0, 100])

plt.ylim([0, 50]) plt.title('Dataset') plt.ylabel('speeding_feature')plt.xlabel('Distance_Feature')

plt.scatter(f1,f2)

plt.show()

# create new plot and data

plt.plot()

colors = ['b', 'g', 'r']

markers = ['o', 'v', 's']

# KMeans algorithm#K = 3

kmeans_model = KMeans(n_clusters=3).fit(X)

plt.plot()

for i, l in enumerate(kmeans_model.labels_):

plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l],ls='None')plt.xlim([0, 100])

plt.ylim([0, 50])plt.show()

Driver_ID,Distance_Feature,Speeding_Feature

3423311935,71.24,28

3423313212,52.53,25
```

3423313724,64.54,27

3423311373,55.69,22

3423310999,54.58,25

3423313857,41.91,10

3423312432,58.64,20

3423311434,52.02,8

3423311328,31.25,34

3423312488,44.31,19

3423311254,49.35,40

3423312943,58.07,45

3423312536,44.22,22

3423311542,55.73,19

3423312176,46.63,43

3423314176,52.97,32

3423314202,46.25,35

3423311346,51.55,27

3423310666,57.05,26

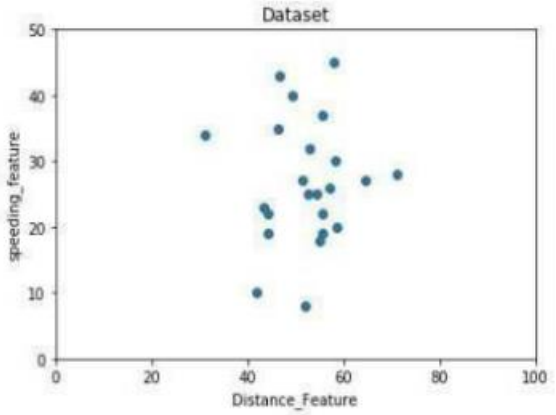
3423313527,58.45,30

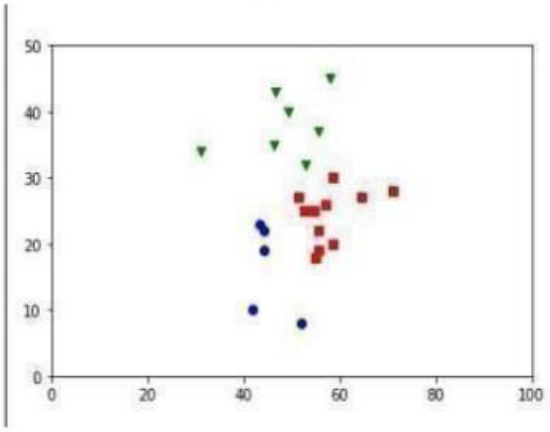
3423312182,43.42,23

3423313590,55.68,37

3423312268,55.15,18

Output:


Dataset
speeding_feature
Distance_Feature
0
10
20
30
40
50
0
20
40
60
80
100


0
10
20
30
40
50
0
20
40
60
80
100

Experiment 11:

Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

```
import csv import random
import math import operator
def loadDataset(filename, split, trainingSet=[] , testSet=[]):with open(filename, 'rb') as csvfile:
lines = csv.reader(csvfile)dataset = list(lines)
for x in range(len(dataset)-1):for y in range(4):
dataset[x][y] = float(dataset[x][y])if random.random() < split:
trainingSet.append(dataset[x])else:
testSet.append(dataset[x])
def euclideanDistance(instance1, instance2, length):distance = 0
for x in range(length):
distance += pow((instance1[x] - instance2[x]), 2)return math.sqrt(distance)
def getNeighbors(trainingSet, testInstance, k):distances = []
length = len(testInstance)-1
for x in range(len(trainingSet)):
dist = euclideanDistance(testInstance, trainingSet[x], length)distances.append((trainingSet[x], dist))
distances.sort(key=operator.itemgetter(1))neighbors = []
for x in range(k):
neighbors.append(distances[x][0])return neighbors
def getResponse(neighbors):classVotes = {}
for x in range(len(neighbors)): response = neighbors[x][-1]if response in classVotes:
classVotes[response] += 1
else:
classVotes[response] = 1
```

```
sortedVotes = sorted(classVotes.iteritems(),reverse=True)
return sortedVotes[0][0]
def getAccuracy(testSet, predictions): correct = 0 for x in range(len(testSet)):
key=operator.itemgetter(1
),
if testSet[x][-1] == predictions[x]:correct += 1
return (correct/float(len(testSet))) * 100.0
def main():
# prepare data trainingSet=[] testSet=[]split = 0.67
loadDataset('knndat.data', split, trainingSet, testSet) print('Train set: ' + repr(len(trainingSet)))
print('Test set: ' + repr(len(testSet)))
# generate predictions predictions=[]k=3
for x in range(len(testSet)):
neighbors = getNeighbors(trainingSet, testSet[x],k) result = getResponse(neighbors)
predictions.append(result)
print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1])) accuracy = getAccuracy(testSet,
predictions)
print('Accuracy: ' + repr(accuracy) +'%') main()
```

OUTPUT

```
Confusion matrix is as follows
[[11 0 0]
[0 9 1]
[0 1 8]]
Accuracy metrics0 1.00 1.00 1.00 11
1 0.90 0.90 0.90 10

2 0.89 0.89 0,89 9
Avg/Total 0.93 0.93 0.93 30
```